An Experiment in Automated Theorem Proving in Type Theory

Fredrik Lindblad and Marcin Benke

Department of Computing Science
Chalmers University of Technology
412 96 Göteborg, Sweden

We present some experiments in automated theorem proving in *Alfa* [HR00], an interactive proof system based on Martin-Löf's type theory [CC99,ML84]. It can be also viewed as a small purely functional programming language with a type system that provides dependent types, thus allowing specification and verification of program properties within its type system. As such, Alfa supports algebraic datatypes, pattern matching and general recursive definitions. To preserve logical consistency, all proofs are subject to termination check as well as typechecking.

Alfa is a term-based proof editor. This means that the proof is presented and recorded as a term, rather than as a tactic expression as in tactic-based proof editors such as *Coq*.

Our tool, given a proposition (type), tries to find its proof (inhabitant). If a solution is found, it is presented as a proof term, which can be verified by the type-checker. The main purpose of the tool is to facilitate interactive proving by filling out the tedious, yet relatively simple details of a proof. One particular application, where such automation is extremely handy, is verification of functional programs.

The main contribution of this work is the use of induction and generalization in automated proofs, which is superior to automation available so far in proof assistants based on higher-order logic, such as the `Auto` tactic in *Coq*.

To ensure termination, we rely on structural induction. However, the tool can construct proofs with nested induction and induction appearing in subproofs as well as generalize (strengthen) the induction hypothesis when needed. Proofs containing case analysis on compound expressions can sometimes be found. This is needed to prove e.g. that the functions *map* and *filter* commute in a certain way.

A notion of quasi normal forms has been investigated. One advantage of this approach is to avoid higher-order unification which is otherwise imposed in type theory. It can also reduce the amount of computation in the search. Although we can miss some equalities this way, tests have indicated that for most problems the fidelity of our notion of equality is sufficient. It also gives useful information as to when case analysis on compound expressions should be performed.

The tool has been tested on multiple examples, mostly related to verification of functional programs. For example, the correctness of the quicksort algorithm has been proved by splitting up the problem in a number of lemmas which can be proved automatically.

References

[CC99] C. Coquand and T. Coquand. Structured type theory. In *Workshop on Logical Frameworks and Meta-languages*, Paris, France, Sep 1999.

[HR00] Thomas Hallgren and Aarne Ranta. An extensible proof text editor. In *Logic for Programming and Automated Reasoning*, volume 1955 of *LNCS*, pages 70–84. Springer, 2000.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984.